ꟼ periodica polytechnica

Electrical Engineering
53/1-2 (2009) 3–9
doi: 10.3311/pp.ee.2009-1-2.01
web: http://www.pp.bme.hu/ee
© Periodica Polytechnica 2009

RESEARCH ARTICLE

Beyond the limits of kinematics in planning keyframed biped locomotion

Tamás Juhász / *Tamás* Urbancsek

Received 2010-05-22

Abstract

Keyframed motion planning is a technique that specifies a robot motion by its joint variable samples in discrete time-steps. In this paper, we aim to provide an off-line (i.e. non real-time) dynamic motion optimizing method for keyframed humanoids. Let's assume that a desired reference movement has been designed, it can be simulated using a real-time kinematics model. Due to dynamic effects the robot segments will not exactly follow the reference trajectories. Assuming a detailed, sophisticated dynamics model (running offline) we can formulate a norm that expresses the difference of dynamic and kinematic simulations. In this article we present our idea, how the motion could be automatically tailored by lowering this norm using numerical methods in a way, that the output of the dynamic model better approximates the reference motion. Finally, we show our experimental results within a modern simulation environment as well as on our test humanoid platform.

Keywords

Humanoid · keyframe · motion planning · dynamics

Acknowledgement

We would like to acknowledge the European Commission and the Fraunhofer Society for the Marie Curie EST Program, furthermore the Hungarian Scientific Research Fund grant No.: T-042634OTKA and the EuroPR Computers Ltd for their kind support in practical exploitation of the research results.

Tamás Juhász
Department Virtual Engineering, Fraunhofer Institute for Factory Operation and Automation,, Sandtorstrasse 22, D-39106 Magdeburg,, Germany
e-mail: Tamas.Juhasz@iff.fraunhofer.de

Tamás Urbancsek
Department of Control Engineering and Information Technology, BME, Magyar tudósok krt. 2, H-1117 Budapest,, Hungary
e-mail: urbi@iit.bme.hu

1 Introduction

The methods of locomotion planning for biped robots have been studied for many years. Presently, several companies have announced the commercial availability of various humanoid robot prototypes with diverse controllability.

However a family of existing (small) humanoids is being controlled in a discrete time manner (using so called "keyframes"). The keyframe technique lets us specify the target angular position of the joints only in discrete time-steps, whereby the length of the individual intervals is also defined. These platforms have built-in path planning algorithms and have their own control electronics: i.e. no external feedback is given about actual joint torques. They do not support the widely spread methods, thus they need a different approach for locomotion planning.

Humanoids should keep their dynamic balance in order not to fall down while walking. Therefore besides geometric motion planning and kinematics, dynamical effects should also be taken into account.

In case of a biped locomotion, start and goal positions are given in a virtual environment. Some existing motion-planner methods (the lazy RPM procedure [2], the randomized planning techniques [3, 5] or the footstep planning [4]) give continuous time functions how to move the robot through the desired path. This means, the reference signal for all the robot joints will be set in every moment.

Albeit, these techniques cannot be used directly for those kinds of systems, which can only be regulated in a discrete time manner externally (using so called "keyframe" inputs). Keyframed humanoids allow the user to specify target servo angles only for discrete moments; their onboard path planner interpolates the reference path for the actual time using n^{th} order polynomial approximations.

In a special case, when $n = 1$, we talk about linear interpolation: the reference path in servo space is a broken line; the reference (angular) speed of the every (rotational) servo remains constant within each of the intervals.

The joint correction signals are mostly produced in a decoupled way by on-board or in-servo controllers and the user has no access to them.

Some systems do not even allow real-time tracking of motion execution; actual joint values cannot be read externally. For these robots the whole motion shall be planned in advance.

This paper deals with these biped platforms and proposes a different approach for locomotion planning.

2 Keyframe based motion planning

2.1 Keyframed character animation

Among the various numerical techniques, the keyframe methodology can also be used to define a robot motion.

Each keyframe includes a snapshot of robot servo parameters and a timestamp that defines the time of sampling. Thereby a robot motion can be defined by a series of keyframes – hereinafter we will refer to them also as motion phases.

Each keyframe includes a snapshot of robot servo parameters and a timestamp that defines the time of sampling. Thereby a robot motion can be defined by a series of keyframes – hereinafter we will refer to them also as motion phases.

2.2 Role of inverse kinematics and inverse geometry in motion planning

The keyframe-based method combined with inverse kinematics makes character animation design faster. The animator defines the new motion phase by altering the previous one. He can move segments both in servo and in Cartesian space: e.g. lifting up left foot or translating body forward. Servo values are computed then by the inverse geometric model. If the animator takes a snapshot of the actual robot state a new motion phase is created.

Using this technique an arbitrary locomotion can be designed. The reference motion is played with help of the kinematic model.

Using this technique an arbitrary locomotion can be designed. The reference motion is played with help of the kinematic model.

2.3 Kinematic model

Let us assume our biped robot has J joints, we will refer to them $1 \leq \mathrm{j} \leq J$. The robot has then $J+1$ segments indexed by $0 \leq \mathrm{j} \leq J$. We can define K keyframes, when we define the τ_k length of the kth time interval ($\rightarrow \tau$ vector) between the phases k and $(k-1)$, as well as the q_{kj} joint angles ($\rightarrow \Theta^{\,[K \times J]}$ matrix).

The robot has a hierarchical structure, where the torso segment plays the role of the root node. If we know all joint angles, we can calculate the local pose of each segment (where "local" means relative to the torso).

As the torso has the segment number j=0, let us call the startup pose of that segment ${}^0\Gamma_0 = {}^j\Gamma_t\,|_{t=0}$ which is a 6 component vector in global Cartesian space as 3 position coordinates and 3 Euler-angles describe the pose of an object in 3D space.

In kinematical modeling it can be assumed that the pose of a robot segment does not change during the motion phase. This so called "unyielding" object can be one of the feet: it stays always on the ground and every segment moves relatively to it. The index of the unyielding segment is also a parameter of each motion phase, so we introduce the $\mathbf{u}$ vector with length k for these indices during the whole motion.

If we know the interpolation method between the keyframes, we can formulate a continuous vector-functional, describing the pose of the jth segment in global Cartesian-space:

$$ {}^j\Gamma^{\mathrm{Kin}}(t) \equiv {}^j\Psi^{\mathrm{Kin}}(\Theta, \mathbf{u}, \tau, {}^0\Gamma_0, t)\,, 0 < j \leq J \qquad (1) $$

The ${}^j\Gamma^{\mathrm{Kin}}(t)$vector has a dimension of 6 $\cdot\Gamma^{\mathrm{Kin}}(t)$ will be used for the aggregate pose vector with 6*$(J+1)$ long representing the pose of every segment:

$$ \Gamma^{\mathrm{Kin}}(t) \equiv \Psi^{\mathrm{Kin}}(\Theta, \mathbf{u}, \tau, {}^0\Gamma_0, t) \qquad (2) $$

From now on, the non-linear operator Ψ^{Kin} will be called the kinematic model of the robot.

3 Iterative motion correction using dynamic model

Let us assume we designed a reference motion using the keyframe-technique: apparently this will run only in the ideal virtual world smoothly. Nevertheless this motion represents exactly what we would like to achieve with the real robot (e.g.: stepping forward by a given step length). If the servos could exactly follow the reference signal, $\Gamma^{\mathrm{Kin}}(t)$ would describe the robot state in Cartesian space. Considering the dynamic effects of the real world, there will be some difference between the real and the reference motions (e.g.: the given forward-step will be shorter or longer) – in extreme case the robot might also fall. We present an algorithm that finds a new motion that approximates $\Gamma^{\mathrm{Kin}}(t)$ better in real world.

3.1 Output of dynamics simulation

As a first step, the dynamic model of our robot has to be built (the general procedure is not included in this article, but details can be found in [7]).

Our motion-optimization process uses an iterative approach, where the Θ input matrix and τ will vary step by step. Using dynamic simulation we have the following output after i steps:

$$ \Gamma^{\mathrm{Dyn}}_{(i)}(t) \equiv \Psi^{\mathrm{Dyn}}\left\{\Theta_{(i)}, \mathbf{u}, \tau_{(i)}, {}^0\Gamma_0\right\}(t) \qquad (3) $$

$\Gamma^{\mathrm{Dyn}}_{(i)}(t)$ is a vector-scalar function of dimension $6\cdot(J+1)$, and it represents the time-dependent pose of all the robot segments, but using the dynamic model for the particular time.

3.2 Iterative conformance enhancement

Taking a given iteration into account, the difference of the actual output of the dynamics simulation and the reference motion is also a vector-scalar function:

$$ \Gamma_{(i)}(t) = \Gamma^{\mathrm{Dyn}}_{(i)}(t) - \Gamma^{\mathrm{Kin}}(t) \qquad (4) $$

In some cases the position error has higher priority than the orientation error; furthermore it is usually desired to have better compliance on the feet as for example on the head. Each segment has a 6 dimensional $\mathbf{w}_j$ pose-weight vector, so for the whole system the following diagonal weight matrix can be introduced:

$$\mathbf{W}(t) = \text{diag} < w_0(t), w_l(t), \dots w_j(t) > \tag{5}$$

This $\mathbf{W}$ matrix has $[(J+1)\cdot 6 \times (J+1)\cdot 6]$ dimensions. In the i$^{\text{th}}$ iteration we can define a weighted error function with help of an inner product:

$$\mathrm{E}^2_{(i)}(t) = < \mathbf{W}\cdot\Gamma_{(i)}(t), \mathbf{W}\cdot\Gamma_{(i)}(t) > \tag{6}$$

Using this error function we want to define aχnorm, in order to have a non-negative scalar value that represents the correspondence between the reference motion and the simulated one (considering the whole simulation length is T):

$$\chi_{(i)}\left\{\Theta_{(i)}, \mathbf{u}, \tau_{(i)}, {}^0\Gamma_0\right\} = \int_0^T \mathrm{E}^2_{(i)}(t)\cdot dt \tag{7}$$

One can easily see that all members $\chi_{(i)}$of the series χdefine a norm for the $\Gamma_{(i)}(t)$ functions, which form a Hilbert space over this norm. As we want to lower this norm, we need to alter the $\Theta_{(i)}$ and $\tau_{(i)}$ input parameters each step, and leave the other input variables constant. During the steps some boundary conditions (initial and final servo configurations, motion duration, etc.) have to be fulfilled.

We would like to have a monotonous descending series of χ_i elements to reach the optimal input keyframes:

$$(\Theta_{\text{opt}}, \tau_{\text{opt}}) = \underset{(\Theta,\tau)}{\text{argmin}}\, \chi\left\{\Theta, \mathbf{u}, \tau, {}^0\Gamma_0\right\} \tag{8}$$

In the next section we present a numerical solution for this problem.

3.3 Numerical solution

The goal of the numerical correction algorithm is that the dynamical behavior approximates the reference movement.

Let us define $\mathbf{D}$ as a subset of input parameter domain $\mathbf{x}_0 := (\Theta_{Kin}, \tau_{Kin})$ with the following properties: $\mathbf{D}$ shall be the largest connected subset that fulfils all the boundary conditions and contains all the motion parameter combinations where the robot does not fall down and contains the reference movement.

Before applying any numerical optimization method the following presumptions must be taken

- $\chi(\mathbf{x})$shall be continuous and differentiable over $\mathbf{D}$,
- close to the boundaries of $\mathbf{D}$ the negative gradient $-\nabla\chi$ points inwards,
- the optimum $\mathbf{x}_{\text{opt}} = (\Theta_{\text{opt}}, \tau_{\text{opt}})$ lies inside of $\mathbf{D}$,
- there are no other local extremities.

Without the exact mathematical proof, we show that in practical cases these presumptions hold

- As long as the robot does not fall down, any infinitesimal change in input parameters effects a proportional infinitesimal change in norm. It is clear that there will be a discontinuity in the norm function where the robot falls.
- Nearby the boundaries of $\mathbf{D}$ the robot motion becomes unstable. The robot body starts to oscillate, therefore the robot segments will have more significant difference from the reference motion, thus the χ norm grows. Consequently, the negative gradient vector points inwards.
- The animator can create a motion, which “seems quite dynamic”; the robot remains standing, and does what he wanted; thus the motion is in $\mathbf{D}$ and close to the optimum we search.

It is clear that the final motion will not match the reference motion. It might be impossible to follow due to ignored dynamics. Therefore, its norm will not reach zero. A local optimum is reached where the gradient is zero.

$\chi(\mathbf{x})$ is a strongly non-linear function of its input parameters with narrow potential tunnels, therefore we decided to implement the non-linear conjugated gradient method described in [8].

For this method we needed the gradient of the potential function. We computed it numerically in a component-by-component way using partial differentials. For the i^{th} component:

$$\nabla\chi(\mathbf{x})_i = \left[\frac{\chi(\mathbf{x}+\Delta\mathbf{x}_i) - \chi(\mathbf{x})}{\Delta\mathbf{x}_i}\right], \Delta\mathbf{x}_i = \pm\varepsilon\cdot\mathbf{e}_i \tag{9}$$

where $\mathbf{e}_i$ is the i^{th} basis vector of the input space. As $\mathbf{D}$ is bounded, it can happen that a $\mathbf{x}+\Delta\mathbf{x}_i$vector points outside of $\mathbf{D}$. It is very uncommon as the negative gradients of $\mathbf{D}$ at the boundaries point inwards, so the algorithm does not approach the boundaries. If it is still the case, one has to take the inverse of $\Delta\mathbf{x}_i$. In an extreme case if it still points outside of $\mathbf{D}$, then $\mathbf{D}$ is very thick along that dimension, most likely the algorithm has reached an extremity of $\mathbf{D}$. The obvious opportunity is to renounce the derivative in this direction at the particular step. In this iteration the dynamical simulation has to be executed $(J{*}K + K{+}1)$ times.

At initial phase, the algorithm has to perform a line search along the direction of steepest descent. It is an iterative method that should find the minimum along this line. It is a one-dimensional search method. The result is $\mathbf{x}_1$.

Then, the algorithm consists of 5 steps:

1 Compute the gradient in the actual position: $\mathbf{x}_n$,

2 Compute β_n according to the Polak–Ribière formula:

$$\beta_n = \max\left[\frac{\nabla\chi(\mathbf{x}_n)^T * (\nabla\chi(\mathbf{x}_n) - \nabla\chi(\mathbf{x}_{n-1}))}{\nabla\chi(\mathbf{x}_{n-1})^T\nabla\chi(\mathbf{x}_{n-1})}, 0\right] \tag{10}$$

3 Compute the next conjugated direction

$$\Lambda x_n = \nabla\chi(\mathbf{x}_n) + \beta_n\Lambda\mathbf{x}_{n-1} \tag{11}$$

4 Perform a line search along the last conjugated direction:

$$\min_{\alpha_n} \chi(\mathbf{x}_n + \alpha_n * \Lambda \mathbf{x}_n) \qquad (12)$$

5 Next iteration will be then

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \alpha_{n*} \Lambda \mathbf{x}_n \qquad (13)$$

The algorithm ends if the gradient sinks below a given threshold. Note that the stability reserve of motion is not guaranteed by the algorithm. It is mainly depending on the reference motion.

4 Implementation

At the Department of Control Engineering and Information Technology of Budapest University of Technology and Economics we have a modified version of a KHR-1 humanoid robot (see Fig. 1), the original of which is a commercial product of Kondo Kagaku Co. Ltd., Japan. This experimental biped platform is 34 centimeters tall, has 21 degrees of freedom, and has an onboard control electronic that can interpret only the aforementioned keyframe-based motions.

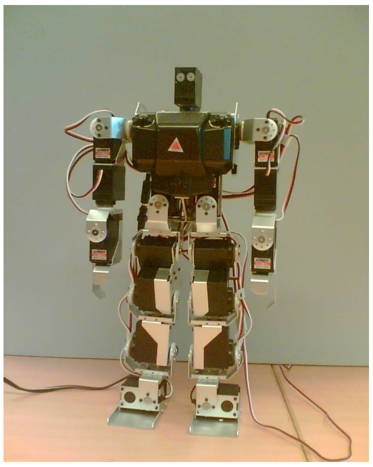

Fig. 1. Our humanoid (KONDO KHR-1)

4.1 Kinematics modeler

We have developed a kinematics-based gait-authoring application for keyframe-controlled robots. This program can be used by an experienced 3D animator to create the keyframes of a desired motion in the virtual world. For this purpose many kinds of interactive tools stay at the user's disposal (a screenshot of the user interface can be seen on Fig. 2).

For each phase of the motion that is currently being edited, one can use the constrained inverse kinematics tools first with the mouse for draft setups, and later fine tune a given group of joints either with the mouse or with the keyboard manually. The length of the individual phases can also be varied, of course.

Assuming our robot is "standing at attention" (we call it as the initial pose), in our example our goal is a single stepping locomotion, where the real robot has to step forward with its left foot exactly 3 centimeters, and then finish movement with the initial pose. We have modeled this locomotion using 9 keyframes (phases), where the startup and the final phase contain exactly the same servo angles. Meanwhile a double support phase (both feet on the floor) transfers to a single support phase (standing on the right foot, swinging the left one), and finally we finish in a double support phase again.

The output of this task is a smooth, harmonic movement in the ideal virtual world (note, that for the time being we neglect the dynamical behavior of the robot). If we play back this motion using a dynamic model, it definitely behaves differently. It is predictable that due to inertial-, contact- and friction forces the real dynamical behavior will issue a pose error at the end, containing two components: a real robot will probably step shorter or longer than 3 cm (position error), and it might deflect from the ideal forward direction (orientation error).

4.2 Using dynamic multibody simulation

In order to overcome the pose error between the realized- and the designed reference motion, we use our presented iterative procedure that reduces this difference. For our method we need a fair dynamic model of the robot.

The Dymola [9] is a multi-engineering modeling and simulation tool, developed by Dynasim AB, Sweden. The multi-engineering capabilities of Dymola present new and revolutionary solutions for modeling and simulation as it is possible to simulate the dynamic behavior and complex interactions between systems of many engineering fields, such as mechanical, electrical, thermodynamic, hydraulic, pneumatic, thermal and control systems. This means that users of Dymola can build more integrated models and have simulations results that better depict reality. Dymola interprets the declarative object-oriented modeling language Modelica [10], and has interfaces to use additional external user modules written in C or FORTRAN languages.

In Dymola we have built a detailed electro-mechanics model (Fig. 3) of the KHR-1 humanoid:

We have developed special building components, which extend the standard models from Modelica.Mechanics.MultiBody library. We had to model contact between objects (and collision response in this manner) and the KRS-784 ICS Digital Servo, which is used in the real KHR-1 robot.

4.2.1 Servo model

The basic actuated revolute joint is encapsulated in a complex servo model (Fig. 4) that contains also the electronic model of the KRS-784. The target angle position control loop is implemented with a simple P controller (the closed loop contains an integrator element, because of the DC motor model). Some parameters of the servo model (e.g.: permanent DC motor's $V_{nominal}$, $I_{nominal}$, R_a, L_a electromagnetic parameters, rotor inertia, gear ratio, nomial rpm speed) can be found in the data-sheet of the KRS-784, and the others are identified after doing some tests.

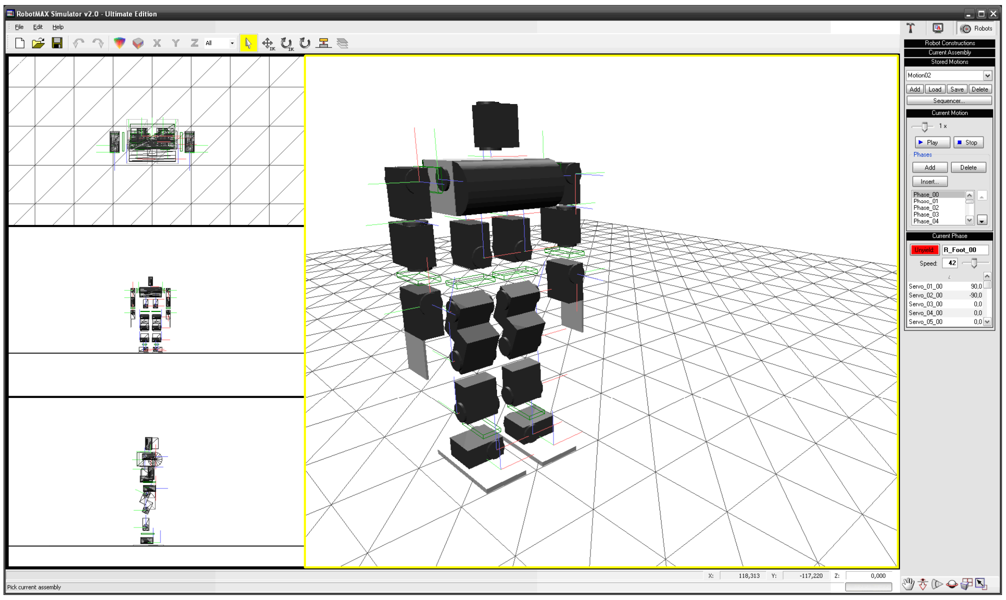

Fig. 2. Main GUI of our kinematics modeler

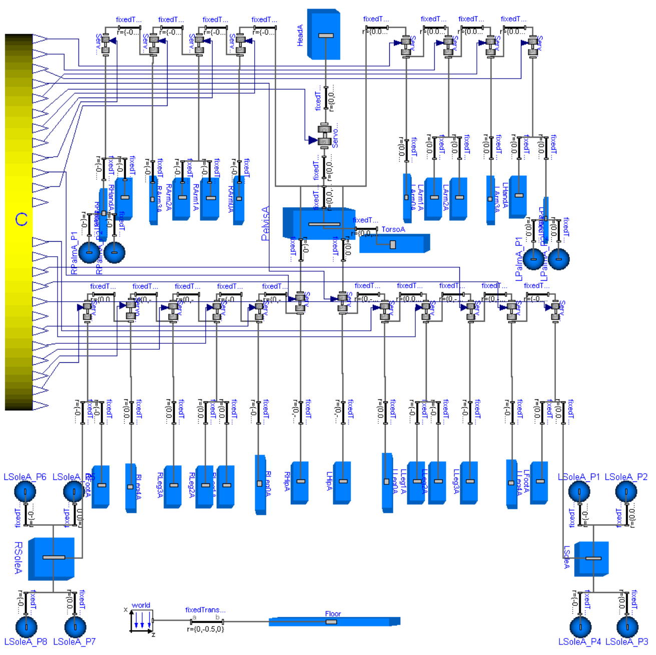

Fig. 3. Our humanoid model in Dymola environment

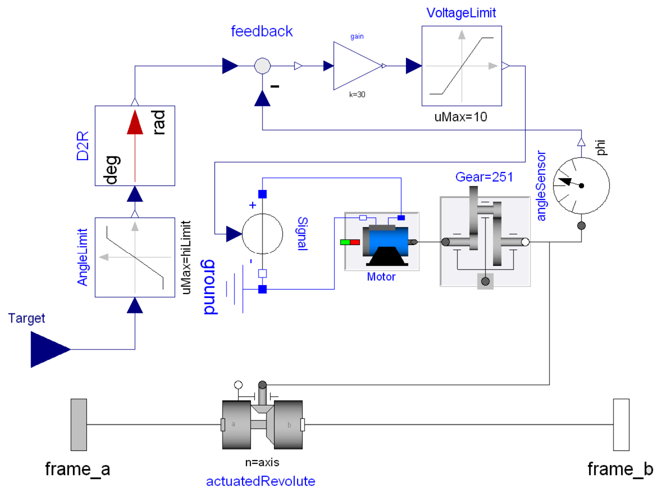

Fig. 4. Our electro-mechanical model of a KRS-784 servo

4.2.2 Contact and collision model

As a part of the Dymola 6.0d environment, the standard Modelica 2.2.1 multibody library does not include contact processing. We had to extend the basic Modelica.Mechanics.MultiBody.Parts.Body rigid body model (containing shape, mass, inertia tensor and the Newton/Euler equations of dynamics) with the support of collision handling.

In an articulated multibody system there are dynamic constraints between the connected rigid bodies. Dymola solves the arisen differential algebraic equations (DAEs) and ordinary differential equation sets (ODEs) internally (partially symbolically), where all state variables – including positions and velocities – have to be differentiable. Thus there is no way to use the other popular impulse-based collision response method, which would require sometimes overriding the objects' velocities instantaneously. This is not allowed in Dymola – because this would make the velocity vectors not differentiable. Because of this, we must use a force based method in collision response.

Besides Modelica language, we used partially external C++ implementation with the popular SOLID interference detection library [11], which can be used to retrieve contact points between pairs of objects (it uses the GJK algorithm [12]), but it does not calculate the response, by default. We made a spring and damper material model, and calculate the contact force in normal direction (along the vector defined by two contact points) the following way:

$$|F_{\mathrm{NORMAL}}| = \begin{cases} 0, \;\; p < 0 \\ \left[1 + \frac{1-\varepsilon}{\varepsilon \cdot v_{\mathrm{COLL}}} \cdot \dot{p}_{\perp}\right] \cdot S \cdot p, \; p \geq 0 \end{cases} \qquad (14)$$

The scalar 'p' means the penetration depth [m]. If we project the relative velocity of the contact points to normal direction vector, we get the signed $\dot{p}_\perp$ [m/s] component of penetration velocity, the value of which is stored in 'v_{COLL}' at the moment of first contact. The spring coefficient 'S' [N/m] and the restitution factor ε of the contacting materials are used in the previous formula.

When the relative velocity of two interpenetrating bodies have nonzero tangential component (v_t), it is very important that we calculate friction forces using a friction model. Without this effort our virtual robot will not be able to make any translational movement at all. These forces are parallel to the plane, the normal of which is the vector between the two contact points. The friction model is the following:

$$|F_{\text{FRICTION}}| = \begin{cases} \mu_{kin} \cdot |F_{\text{NORMAL}}|\,, & |v_t| > v_{st} \\ \mu_{stat} \cdot \frac{|v_t|}{v_{st}} \cdot |F_{\text{NORMAL}}|\,, & |v_t| < v_{st} \end{cases} \tag{15}$$

We have two friction coefficients for the static and for the kinetic cases. The constant speed value v_{st} represents the limit, which influences whether objects are considered as sliding or stay resting. The result of these two forces will act in opposite directions on both objects in each colliding pair.

4.2.3 Implementing our iterative enhancement method for the realized motion

We implemented the iterative algorithm in Matlab environment. Fig. 5 explains the block scheme of the implementation, with the three main software components:

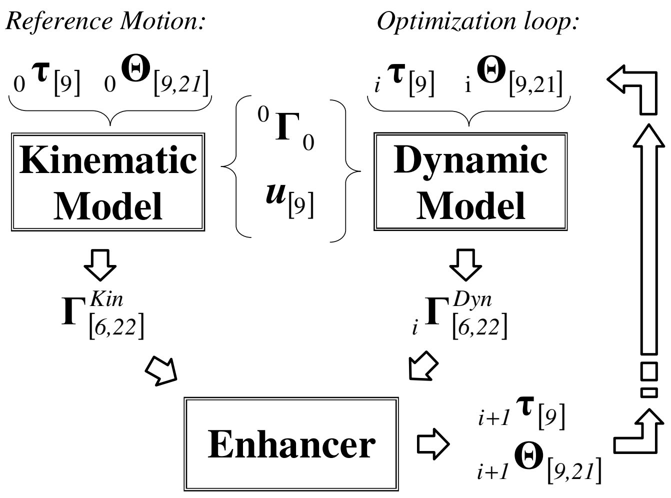
Reference Motion:
Optimization loop:
${}_0\boldsymbol{\tau}_{[9]}$ ${}_0\boldsymbol{\Theta}_{[9,21]}$
${}_i\boldsymbol{\tau}_{[9]}$ ${}_i\boldsymbol{\Theta}_{[9,21]}$
Kinematic Model
${}^0\boldsymbol{\Gamma}_0$
$\boldsymbol{u}_{[9]}$
Dynamic Model
$\boldsymbol{\Gamma}^{Kin}_{[6,22]}$
${}_i\boldsymbol{\Gamma}^{Dyn}_{[6,22]}$
Enhancer
${}_{i+1}\boldsymbol{\tau}_{[9]}$
${}_{i+1}\boldsymbol{\Theta}_{[9,21]}$

Fig. 5. The three main modules of the implemented algorithm

The "Kinematic Model" module serves the reference pose-time functions of all robot segments using linear keyframe interpolation. The "Enhancer" module can query the pose of a given segment at any time instant between 0 and T (the simulation length).

In all iterations the "Dynamic Model" module calls a Dymola block that calculates actual pose functions for the segments, using the created dynamic model. The Enhancer analyses the difference of these outputs and calculates the new input keyframes (joint angles and interval-lengths) according to the method presented in sections 3.2 and 3.3. The output of the Enhancer is fed back to the Dynamic Model, thus forming a closed loop of iterative motion enhancement procedure.

5 Results and applications

We tested our locomotion enhancement algorithm on several stepping motions. In four particular cases the kinematic model prescribed a step forward motion starting with the right foot and with various lengths (e.g.: 2.5, 3, 4 and 5 cm). Using the dynamic model the robot stepped askew and turned about 7-11 degrees right.

The components of the $\mathbf{w}_i$ weight vectors in equation (5) were always set to identity for the position coordinates and 0.1 for the Euler-angles, so that angular errors were punished equally to the arc length of a 10 cm long section (average height of center of gravity). The duration of the simulation (T) was set to 1.5 times longer than the last motion phase timestamp, in order to incorporate the analysis of the oscillations at the end of the motions that should also be decayed.

Fig. 6 shows the iterated norm function in the four aforementioned particular cases. The horizontal axis shows the iterations taken by the enhancer algorithm. As the reference motions have 9 keyframes and the robot has $J = 21$ joints, the dimension of the optimization problem were $9\cdot(21+1)=198$ in these examples. This dimension equals also the number of complete iterations that were running each time.

In order to normalize the vertical axis of Fig. 6, the result norm values were divided here by their initial (t=0) values, therefore each motion has a unit norm before the first iteration.

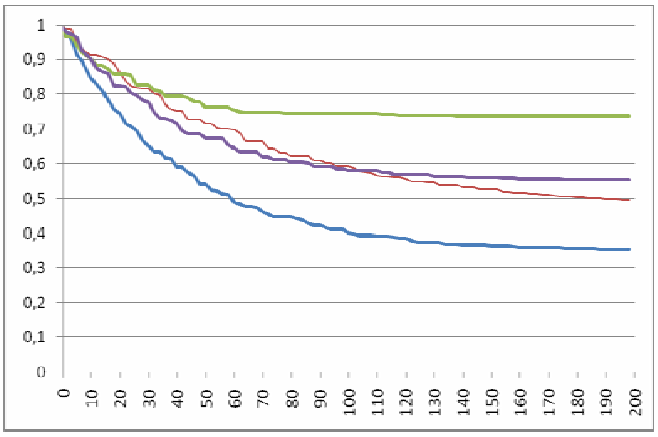

Fig. 6. The norm function during the optimization of four locomotions

It can be seen that such minima have been found for each reference motion, where the monotonous decreasing norm cannot be reduced further. In the various cases the initial norm could be lowered by 25-65 percent, so the matching of the reference motion related to the dynamic simulation was improved significantly.

This motion correction algorithm runs very long, however the demanded optimization of the reference motions doesn't need to be real-time. In Dymola the dynamic model of the humanoid can run with a speed almost half the real-time on a 3 GHz Intel Pentium 4 processor. The reference keyframed motions used in our examples were 1.5 s long; Dymola requires ∼4.5 s for each run of the dynamic model. Computation of the gradient (partial differences along each dimension) requires 199 simulation runs – so it takes about 15 minutes. In addition the line search method (13) requires 40 additional iterations, thus a single enhancement cycle requires ca. 18 minutes. Our implemented nonlinear conjugate gradient method requires at least as many complete iterations as the dimensionality of the problem (198 in the examples above). Therefore the total optimization took approximately 59 hours for a single motion.

The actual contact processing implementation introduces rapidly changing state variables in the system. This makes the ordinary differential equation set stiff, so the variable step-length ODE-solver of Dymola has to take sometimes extreme small iteration steps. The whole implementation of our algorithm is now a single-threaded process, so the main steps are carried out strictly sequentially.

Despite its high time complexity our new algorithm can result smoother motions, so the real robots behave similarly to the reference. Thanks to the well-designed reference motions, the results are also robust to model parameter changes.

6 Future work

The generated reaction forces in 4.2.2 can be smoothed with a polynomial in order to make the ODE better conditioned, so the solver can take larger steps (thus simulation can run faster). Sometimes – because of numerical precision problems – the algorithm has to be reset to use rather the gradient direction instead of following a line-search along the computed conjugate direction.

Calculating the gradient vector in the first main step of the algorithm requires numerical differentiation of a multi-dimensional function. Using modern parallel computation systems (multi-core systems, computer clusters) this task could be easily parallelized, so the algorithm could be considerably accelerated.

References

1 **Amato N, Bayazit O, Dale L, Jones C, Vallejo D**, *Choosing good distance metrics and local planners for probabilistic roadmap methods*, IEEE Trans. Robot. & Autom. **16** (August 2000), no. 4, 442–447.

2 **Bohlin R, Kavraki L**, *Path planning using Lazy PRM*.

3 **LaValle SM, Kuffner JJ**, *Randomized kinodynamic planning*, Proc. IEEE Int Conf. Robot. & Autom. (ICRA), May 1999, DOI 10.1109/ROBOT.1999.770022, (to appear in print).

4 **Kuffner JJ, Nishiwaki K, Kagami S, Inaba M, Inoue H**, *Footstep planning among obstacles for biped robots*, Proc. IEEE/RSJ Int. Conf. Intell. Robot. & Sys. (IROS), October 2001, DOI 10.1109/IROS.2001.973406, (to appear in print).

5 **Kuffner JJ, Kagami S, Nishiwaki K, Inaba M, Inoue H**, *Dynamically-stable motion planning for humanoid robots*, Autonomous Robots (special issue on Humanoid Robotics) **12** (2002), no. 1, 105–118.

6 **Girard M**, *Interactive design of computer-animated legged animal motion*, IEEE Computer Graphics & Applications **7** (June 1987), no. 6, 39–51, DOI 10.1145/319120.319131.

7 **Vukobratovic M, Borovac B, Surla D, Stokic D**, *Biped Locomotion: Dynamics, Stability, Control, and Applications*, Springer-Verlag, Berlin, 1990.

8 *Nonlinear conjugate gradient method – Wikipedia [2007.05.21]*, available at `http://en.wikipedia.org/wiki/Nonlinear_conjugate_gradient`.

9 *Dynasim AB: Dynamics Modeling Laboratory*, available at `http://www.dynasim.com`.

10 *Modelica*, available at `http://www.modelica.org`.

11 *SOLID - Software Library for Interference Detection*, available at `http://www.dtecta.com`.

12 **Gilbert EG, Johnson DW, Keerthi SS**, *A fast procedure for computing the distance between complex objects in three-dimensional space*, IEEE Journal of Robotics and Automation **4** (1988), no. 2, 193–203.